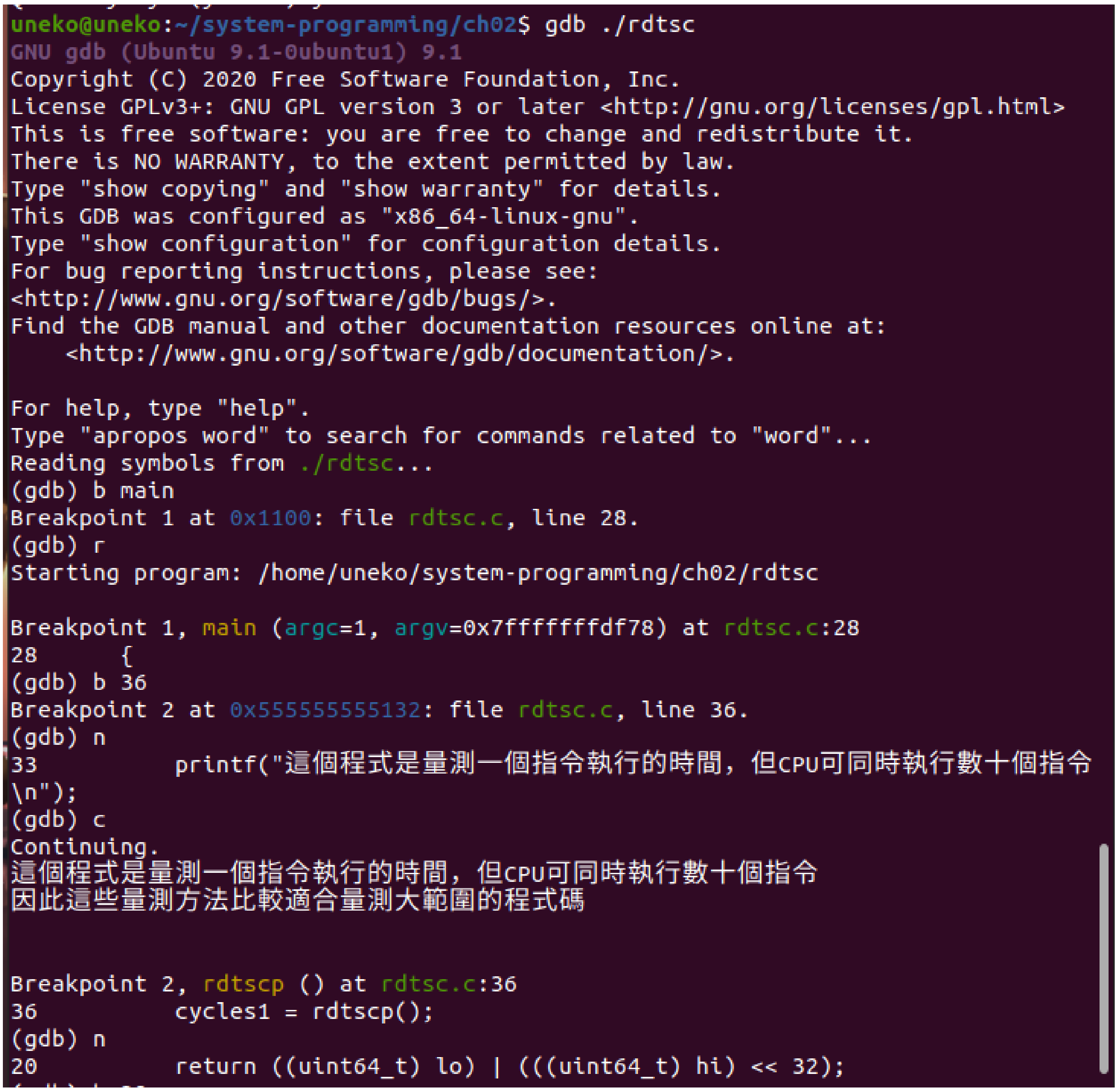
作業內容:

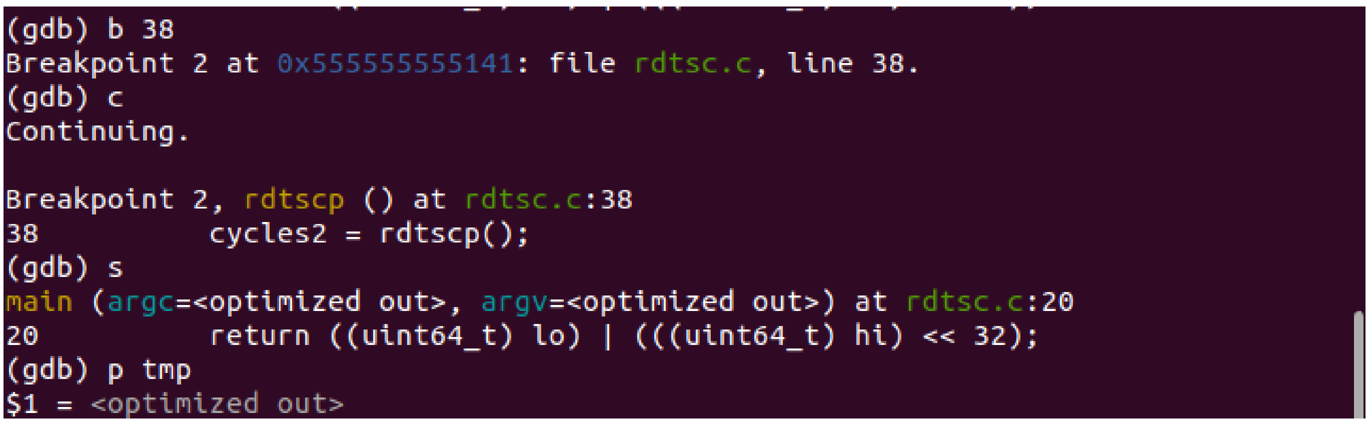
使用gdb對rdtsc.c除錯



```
uneko@uneko:~/system-programming/ch02$ gdb ./rdtsc
GNU gdb (Ubuntu 9.1-0ubuntu1) 9.1
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./rdtsc...
(gdb) b main
Breakpoint 1 at 0x1100: file rdtsc.c, line 28.
(gdb) r
Starting program: /home/uneko/system-programming/ch02/rdtsc

Breakpoint 1, main (argc=1, argv=0x7fffffffdf78) at rdtsc.c:28
28      {
(gdb) b 36
Breakpoint 2 at 0x555555555132: file rdtsc.c, line 36.
(gdb) n
33              printf("這個程式是量測一個指令執行的時間，但cpu可同時執行數十個指令
\n");
(gdb) c
Continuing.
這個程式是量測一個指令執行的時間，但cpu可同時執行數十個指令
因此這些量測方法比較適合量測大範圍的程式碼


Breakpoint 2, rdtscp () at rdtsc.c:36
36              cycles1 = rdtscp();
(gdb) n
20              return ((uint64_t) lo) | (((uint64_t) hi) << 32);
```

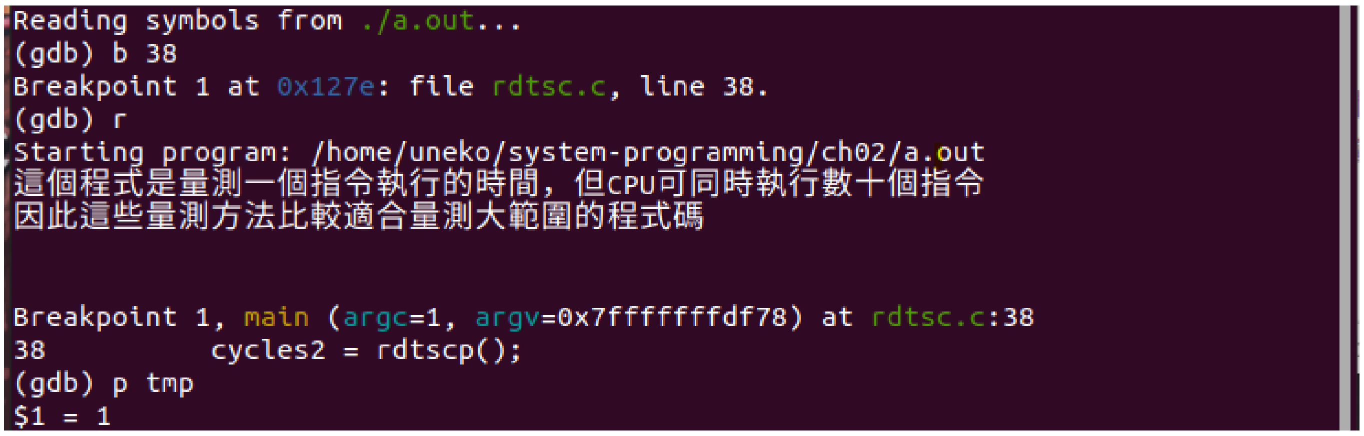
操作內容(1)

1. gdb ./rdtsc 對執行檔rdtsc進行gdb除錯
2. b main 設定中斷點在main
3. r 開始執行
   →執行到main函式停止
4. b 36 設定中斷點在36行
5. n 單步執行(遇函數不進入)
   →執行到33行(出現函式printf)停止
6. c 繼續執行
   →執行到36行(breakpoint)停止
7. n單步執行(遇函數不進入)
   →執行到rdtscp函數的return函式


```
(gdb) b 38
Breakpoint 2 at 0x555555555141: file rdtsc.c, line 38.
(gdb) c
Continuing.

Breakpoint 2, rdtscp () at rdtsc.c:38
38          cycles2 = rdtscp();
(gdb) s
main (argc=<optimized out>, argv=<optimized out>) at rdtsc.c:20
20          return ((uint64_t) lo) | (((uint64_t) hi) << 32);
(gdb) p tmp
$1 = <optimized out>
```

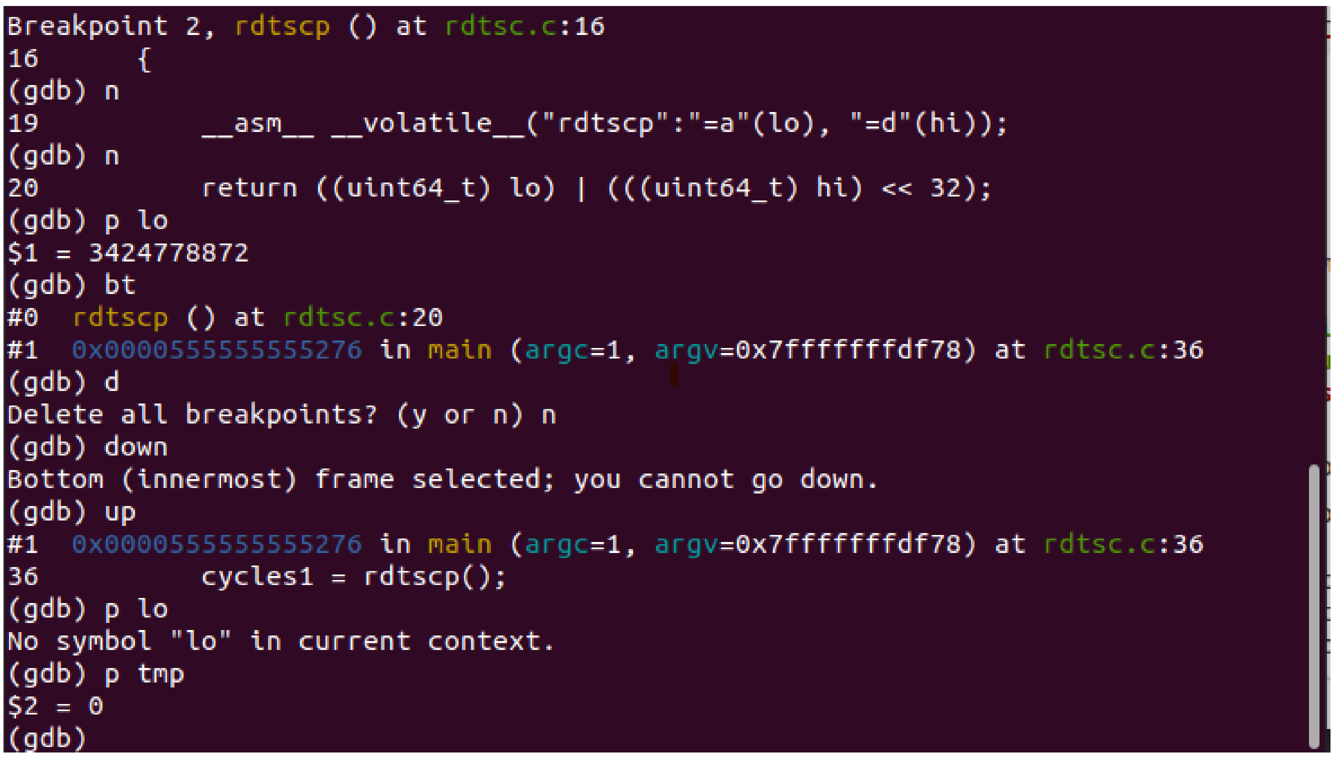
操作內容(2)

1. b 38 設定中斷點在38行
2. c 繼續執行
   →執行到38行(breakpoint)停止
3. s 單步執行(遇函數進入)
   →執行到rdtscp函數的return函式
4. p tmp 印出tmp的內容
   →顯示<optimized out> (表示編譯時因優化(-O3)而未將此變數放入記憶體)


```
Reading symbols from ./a.out...
(gdb) b 38
Breakpoint 1 at 0x127e: file rdtsc.c, line 38.
(gdb) r
Starting program: /home/uneko/system-programming/ch02/a.out
這個程式是量測一個指令執行的時間，但CPU可同時執行數十個指令
因此這些量測方法比較適合量測大範圍的程式碼


Breakpoint 1, main (argc=1, argv=0x7fffffffdf78) at rdtsc.c:38
38          cycles2 = rdtscp();
(gdb) p tmp
$1 = 1
```

操作內容(3)

1. gcc -g rdtsc.c 以不進行優化處理方式編譯
2. p tmp 印出tmp的內容
   →此時可以顯示出變數tmp的內容

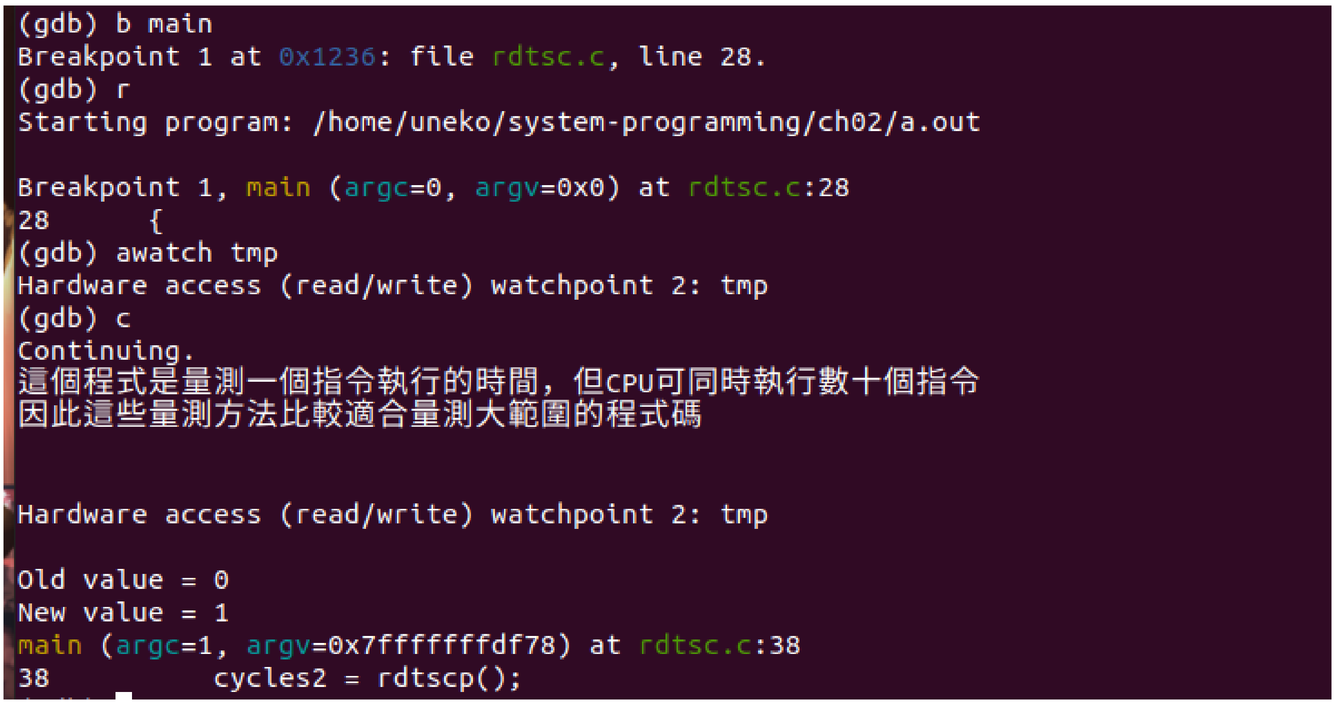


```
Breakpoint 2, rdtscp () at rdtsc.c:16
16      {
(gdb) n
19              __asm__ __volatile__("rdtscp":"=a"(lo), "=d"(hi));
(gdb) n
20              return ((uint64_t) lo) | (((uint64_t) hi) << 32);
(gdb) p lo
$1 = 3424778872
(gdb) bt
#0  rdtscp () at rdtsc.c:20
#1  0x0000555555555276 in main (argc=1, argv=0x7fffffffdf78) at rdtsc.c:36
(gdb) d
Delete all breakpoints? (y or n) n
(gdb) down
Bottom (innermost) frame selected; you cannot go down.
(gdb) up
#1  0x0000555555555276 in main (argc=1, argv=0x7fffffffdf78) at rdtsc.c:36
36              cycles1 = rdtscp();
(gdb) p lo
No symbol "lo" in current context.
(gdb) p tmp
$2 = 0
(gdb)
```

操作內容(4)

進入rdtscp( )

1. **p lo** 執行至第20行後印出lo變數內容
2. **bt** 顯示目前所在函數(rdtscp())及其前函數(main())
3. **d** 刪除breakpoint
4. **up down** 進入跳出函數
5. 跳出rdtscp( ) 後找不到lo變數因此無法印出lo內容

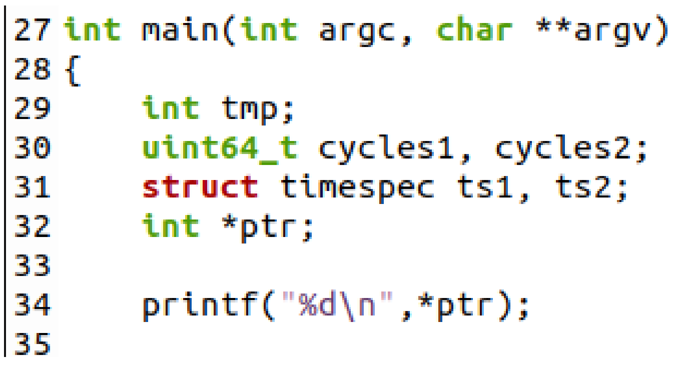


```
(gdb) b main
Breakpoint 1 at 0x1236: file rdtsc.c, line 28.
(gdb) r
Starting program: /home/uneko/system-programming/ch02/a.out

Breakpoint 1, main (argc=0, argv=0x0) at rdtsc.c:28
28      {
(gdb) awatch tmp
Hardware access (read/write) watchpoint 2: tmp
(gdb) c
Continuing.
這個程式是量測一個指令執行的時間，但CPU可同時執行數十個指令
因此這些量測方法比較適合量測大範圍的程式碼


Hardware access (read/write) watchpoint 2: tmp

Old value = 0
New value = 1
main (argc=1, argv=0x7fffffffdf78) at rdtsc.c:38
38              cycles2 = rdtscp();
```

操作內容(5)

使用awatch觀測變數變化

1. **awatch tmp** 將tmp放入watchpoint
2. **"Old value = 0" " New value = 1"**
   執行continue後發現tmp變化並停在變化後的行數

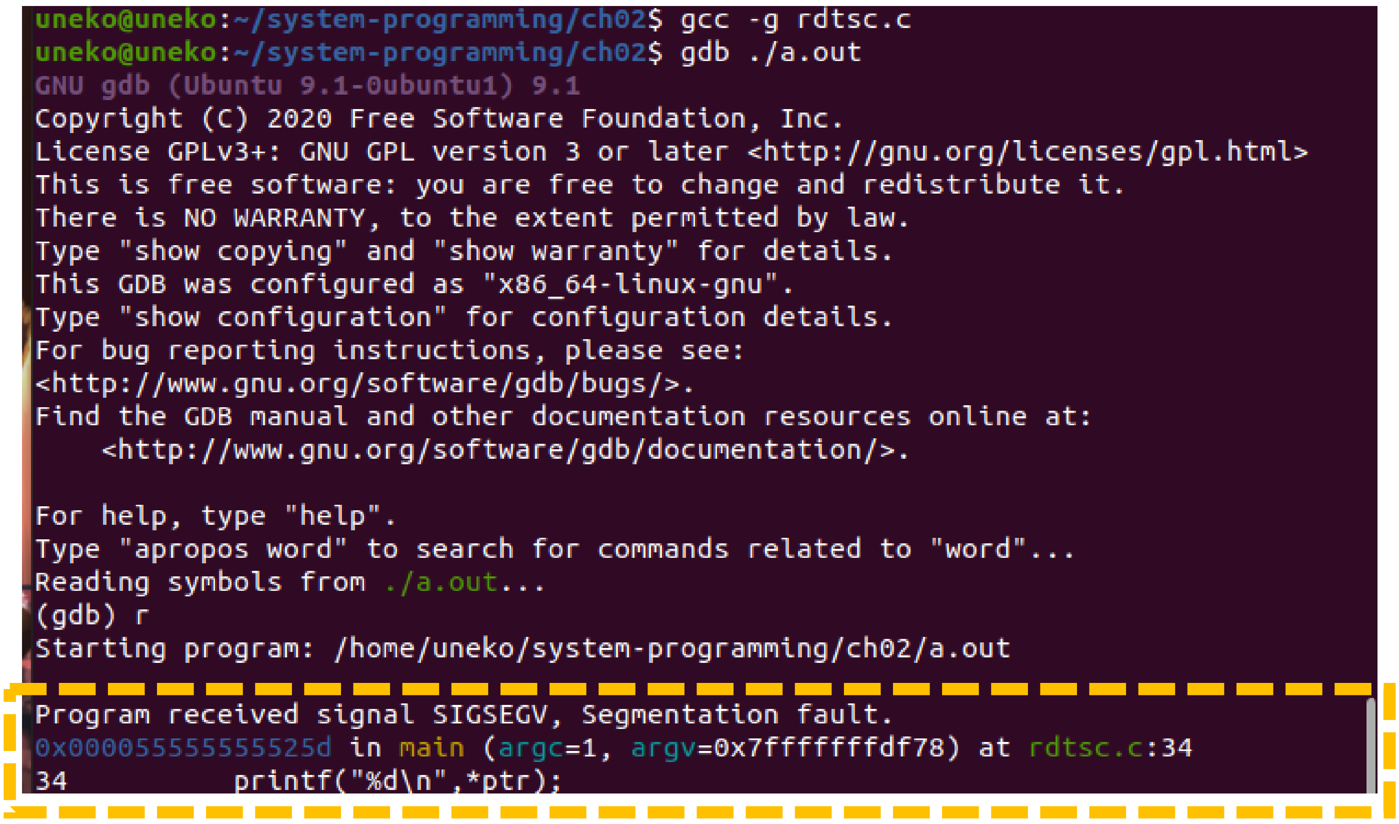


```
27 int main(int argc, char **argv)
28 {
29     int tmp;
30     uint64_t cycles1, cycles2;
31     struct timespec ts1, ts2;
32     int *ptr;
33 
34     printf("%d\n",*ptr);
35 
```

操作內容(5)

修改程式碼內容，故意存取錯誤記憶體

1. int *ptr 宣告指標變數ptr並未指定其位址
2. printf(“%d\n” ,*ptr); 印出ptr位址



```
uneko@uneko:~/system-programming/ch02$ gcc -g rdtsc.c
uneko@uneko:~/system-programming/ch02$ gdb ./a.out
GNU gdb (Ubuntu 9.1-0ubuntu1) 9.1
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./a.out...
(gdb) r
Starting program: /home/uneko/system-programming/ch02/a.out

Program received signal SIGSEGV, Segmentation fault.
0x000055555555525d in main (argc=1, argv=0x7fffffffdf78) at rdtsc.c:34
34          printf("%d\n",*ptr);
```

操作內容(6)

修改程式碼後再重新編譯一次並進入gdb模式執行
→出現 seg. fault 並顯示錯誤行數